Strategies for Incorporating Delegation into Attribute-Based Access Control (ABAC)

Daniel Servos and Sylvia L. Osborn

The University of Western Ontario
London, Ontario, Canada, N6A 5B7
dservos5@uwo.ca, sylvia@csd.uwo.ca

Abstract. Attribute-Based Access Control (ABAC) is an emerging model of access control that has gained significant interest in both recent academic literature and industry application. However, to date there have been almost no attempts to incorporate the concept of dynamic delegation into ABAC. This work lays out a number of possible strategies for incorporating delegation into existing ABAC models and discusses the potential trade-offs associated with each strategy. Delegation strategies are categorized into families that share a number of similar properties. It is our hope that this preliminary work will aid in future ABAC based delegation research by identifying and detailing the challenges and opportunities intrinsic to each method of integrating delegation.

1 Introduction

Attribute-Based Access Control (ABAC) is a relatively new form of access control that bases access control decisions on the attributes of users, objects and the environment rather than the identity of users or the roles/clearances assigned to them. While there has been significant interest in the creation, enforcement and application of ABAC models[5, 7], to date there are few works that address how delegation might be implemented or supported.

Delegation enables a user to temporarily and dynamically alter the design of an access control system after policies have been created to account for everyday changes that policies are insufficient to address. In traditional models of access control delegation is relatively straightforward. A set of permissions or a role membership is delegated directly by a delegator to a delegatee under set conditions (e.g. an expiry date). In ABAC, this is complicated by both the introduction of attributes and ABAC's identity-less nature (i.e. access decisions are made on the basis of attributes and the user's identity may be unknown). Attributes may seem like an ideal access control element to build delegation around (as is done in ABE[2, 6] and Attribute Certificates[8]); however, as we will show, this naive approach comes with a number of unexpected challenges.

This paper offers a preliminarily investigation into strategies for incorporating delegation into ABAC. Potential strategies are created by evaluating the combinations of delegators, delegatable access control elements and delegatees common in most ABAC models (Sect. 2.1). The trade-offs associated with each

family of strategies are discussed and multiple examples are given that demonstrate how delegation might be performed (Sect. 2.2). Finally, we give conclusions and outline directions for future work (Sect. 3). It is our hope that this work will aid future research by identifying possible strategies for the creation of ABAC delegation models as well as the challenges and benefits associated with them.

2 Strategies for Incorporating Delegation in ABAC

2.1 Delegation Components

Delegation can be thought of as relating three access control components; a delegator, a delegatee and a delegatable access control element. A delegator temporarily grants a delegatee an access control element (e.g. a set of permissions or role membership) under set constraints. In RBAC delegation models, this is relatively straightforward: the delegator and delegatee are typically users and the access control element being delegated is either a set of permissions (via a temporary role)[9] or membership in an existing role[1]. ABAC, however, presents new possibilities for delegators, delegatees and delegatable elements that result in different trade-offs and limitations when combined. Each combination provides a conceivable strategy for delegation and offers particular advantages/disadvantages if used as the basis for an ABAC delegation model.

Delegatable elements are the most important characteristic of delegation as they answer *what* is being delegated, while the delegators and delegatees answer *who* and *where* (i.e. *who* is doing the delegating and *where* the elements are being delegated to). The following are the most suitable delegatable elements that we have identified in current ABAC models[5, 7, etc.]:

Attributes: Perhaps the most obvious element and one that has been explored to a limited extent (in ABE[2, 6] and Attribute Certificates[8]) are user attributes. In cases where attributes are delegatable, users are allowed to delegate their assigned attributes to a delegatee such that they are considered to be part of the delegatee's attribute set.

Permissions: Delegating permissions a delegator has obtained from a policy decision is another option. In such cases users are granted permissions as a result of their attribute set satisfying a policy and can delegate these permissions onto others while the policy remains satisfied.

Group Membership: Recent ABAC models have incorporated the concept of user groups into the core ABAC model. In HGABAC[7], groups can be directly assigned user attributes that are inherited by users through their membership. Membership in these groups provides a possible delegatable element, similar to how role membership is delegatable in some RBAC delegation models[1].

While traditional models focus on delegation between users, additional possibilities exist for ABAC. In ABAC models with group support, user groups can be delegators in the sense that attributes or other delegatable elements assigned to groups may be temporarily delegated to a delegatee. In such a case, while the group is the source of the delegatable elements, the actual instigator of the delegation would be the members of the group or another actor in the system (e.g. a

Table 1. Delegation Strategies

Strategy Name	X	DE	Y
Attribute Delegation			
User-to-User Attribute Delegation	U	AS	U
User-to-Group Attribute Delegation	U	AS	G
Group-to-Group Attribute Delegation	G	AS	G
Group-to-User Attribute Delegation	G	AS	U
User-to-Attribute Attribute Del.	U	AS	A
Group-to-Attribute Attribute Del.	G	AS	A
User-to-Policy Attribute Delegation	U	AS	P
Group-to-Policy Attribute Delegation	G	AS	P
Group Membership Delegation			
User-to-User Membership Delegation	U	GM	U
Group-to-User Membership Del.	G	GM	U
Group-to-Group Membership Del.	G	GM	G
User-to-Group Membership Del.	U	GM	G
User-to-Attribute Membership Del.	U	GM	A
Group-to-Attribute Membership Del.	G	GM	A
User-to-Policy Membership Delegation	U	GM	P
Group-to-Policy Membership Del.	G	GM	P

Strategy Name	X	DE	Y
Permission Delegation			
User-to-User Permission Delegation	U	PS	U
User-to-Group Permission Delegation	U	PS	G
Group-to-User Permission Delegation	G	PS	U
Group-to-Group Permission Del.	G	PS	G
User-to-Attribute Permission Del.	U	PS	A
Group-to-Attribute Permission Del.	U	PS	A
User-to-Policy Permission Delegation	U	PS	P
Group-to-Policy Permission Del.	G	PS	P

Legend

U = User
G = Group
P = Policy
A = Attribute
PS = Policy Set
AS = Attribute Set
GM = Group Membership
X = Delegator
DE = Delegatable Element
Y = Delegatee

group leader). Similarly, the delegatee need not be limited to a user. Delegating to a group allows a delegator to assign their delegatable elements to multiple users in one operation. This is useful in scenarios where multiple users are briefly required to take on the duties of a single delegator (e.g. an absent store manager delegating his permissions to all department managers). In cases where group membership is being delegated, it can be considered that all members in the delegatee group are also temporarily made members of the delegated group.

Delegations can also be made to a policy or attribute. When an attribute is acting as a delegatee, all users that are directly (not through delegation) assigned the same attribute also become delegatees. For example if a permission, P, is delegated to the attribute *(ROLE, {manager})* (an attribute named *ROLE* with the value "manager") all users that are assigned the attribute *ROLE* with a value of "manager" will be delegated the permission P. Using a policy as a delegatee works similarly. A delegator delegates some element to a policy they create and all users satisfying this policy are delegated the element. For example, if membership in a group, G, is delegated to the policy "*ROLE = manager AND YEARS_EMPLOYED* $\geq$ *3*", users that have attributes stating that they are managers and employed for at least 3 years will be delegated membership in group G. While delegating to an attribute or policy may seem complex, it is a necessity to support delegation in a system where the identity of a user may remain unknown and access decisions are made purely on the user's attributes.

2.2 Delegation Strategies

Each delegation components described in Sect. 2.1 may be combined to create a delegation strategy. For example the combination *(Users, Permissions, Users)* represents a strategy in which users can delegate their permissions to other users, whereas *(Groups, Attributes, Policies)* would be a strategy in which groups can delegate their attributes to any user that satisfies a policy. Table 1 categorizes each strategy into families based on the element being delegated. Strategies in the same family tend to share common characteristics and challenges for systems adopting them. In this section, we discuss the advantages and limitations of each family. It is assumed that only one strategy is used at a time. While hybrid strategies are possible, and could offer advantages, they are left to future work.

Attribute Delegation In Attribute Delegation strategies, delegatees are delegated a subset of the delegator's attributes. Delegated attributes are merged with the delegatee's directly assigned attributes (i.e. assigned through any means but delegation) and the combined attribute set is treated as the delegatee's set during policy evaluation. An example of User-to-User Attribute Delegation is shown in Fig. 1 where $direct(user)$ is the user's directly assigned attributes and $effective(user)$ is the user's effective attributes (i.e. the merged attribute set used for policy evaluations). In Fig. 1, Alice wants to delegate a subset of her attributes to a prospective student (Dave) so he can satisfy the policy *"role = "undergrad" AND year ≥ 2"* to view some resource. As Dave only has the value "ProspectiveStudent" for his *role* attribute and no *year* attribute, Alice must delegate both her *role* and *year* attributes for Dave to satisfy the policy. The subset Alice delegates is {*(year, {4}), (role, {"undergrad"})*} which makes Dave's effective attribute set {*(role, {"ProspectiveStudent", "undergrad")*}, *(year, {4})*}.

Multiple simultaneous delegations to a single user are also possible. In Fig.1, Alice wishes to delegate to Charlie so he can satisfy the policy *"role IN {"undergrad", "grad"} AND department = "CompSci""*, and access a resource limited to CompSci students. At the same time, Bob wishes to delegate to Charlie so he can satisfy the policy *"role = "faculty" AND department = "SoftEng""* and access a resource limited to SoftEng faculty. Alice delegates {*(department, {"CompSci"})*} and Bob {*(role, {"faculty"})*}. Making Charlie's effective attributes {*(role, {"grad", "faculty"}), (department, {"SoftEng", "CompSci"})*}.

While this style of delegation is easy to implement (a subject's effective attribute set is simply used in place of their direct set), it can lead to serious problems if not carefully constrained. The first issue is the creation of conflicting policy evaluations. In Fig. 1 Alice's delegation results in Dave's effective attribute set containing two values for the *role* attribute, "ProspectiveStudent" and "undergrad". If a policy were to exist such as *"role ≠ "ProspectiveStudent""* two different results would be possible depending on the value of *role* used when evaluating the policy. A potential solution is to use a policy language that specifies clear resolutions to conflicts (e.g. prioritize attributes assigned via delegation over those directly assigned or always grant access when any combination of attributes satisfies the policy). However, the issue is further complicated when

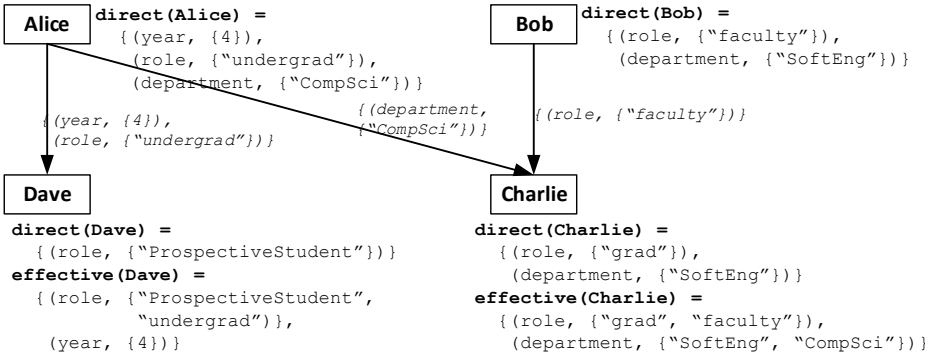

Fig. 1. Example of User-to-User Attribute Delegation. Arrows denote direction of delegation (arrow points to delegatee), boxes represent users of the system.

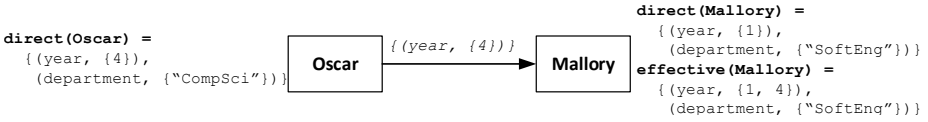

Fig. 2. Example of a possible attack on User-to-User Attribute Delegation.

multiple delegations to the same delegatee are considered simultaneously. In such cases, conflicts can arise from purely delegated attributes, making conflict handling more difficult (e.g. can not simply prioritize delegated attributes).

A second issue is the potential for users to collude to satisfy a policy that they would individually be unable to. In Fig. 2 Oscar and Mallory are trying to satisfy the policy *"year > 2 AND department = "SoftEng""*. Individually, neither can satisfy the policy as Oscar lacks a *department* attribute with a "SoftEng" value and Mallory lacks a *year* attribute with a value greater than 2. However, if Oscar delegates {*"year", {4}*} to Mallory it creates the effective attribute set {*(year, {1, 4}), (department, {"SoftEng"})*} and Mallory can satisfy the policy if *year* is evaluated as 4. While one solution is to heavily constrain what attributes can be delegated or to use a constraint specification language[3] to enforce SoD style constraints, the simplest fix is to isolate delegated attribute sets from each other and the delegatee's directly assigned set. Thus, a user must choose what set of attributes to activate at the start of a session (similar to role activation in RBAC[4]). Isolation of attribute sets would also provide a solution to conflicting policy evaluations and aid in user comprehension. For example, Alice would know that if she delegates all of her attributes to Dave, at most Dave would have access to the same permissions as he did before in addition to the permissions Alice has access to. Users would still be able to bypass negative polices like *"year ≠ 4 AND year ≠ 1"* if not having a *year* attribute is considered to satisfy the policy by delegating a subset of their attributes that omits the *year* attribute.

A third issue resulting from merging attribute sets is losing the descriptiveness of the delegatee's attributes. In Fig. 1, after delegation, Dave's effective attribute set is no longer descriptive of Dave. Dave obtains a *year* attribute with a value of 4 while not being a student. While this makes delegation possible and allows Dave to satisfy the policy, it complicates policy creation (need to account for unexpected attribute combinations) and restricts the use of attributes to the purpose of access control (e.g. a system could not trust that an e-mail sent to an address in a user's effective attribute set was actually theirs).

The last issue is comprehension of what is being delegated and what needs to be delegated to achieve a desired result. A delegator must be familiar with the policies of the system and their own attributes. In Fig. 1, if Alice wanted to delegate a permission she was granted from satisfying the policy *"role = "undergrad" AND year ≥ 2"* she would have to understand the policy, what attribute set she has been assigned and what attribute subset to delegate. This is further complicated if delegated attribute sets are not isolated, as Alice would also have to be aware of possible conflicts and unexpected attribute combinations.

Group Membership Delegation Group Membership Delegation requires an ABAC model which supports user groups in which members of a group inherit attributes assigned to that group. Fig. 3 shows an example of how user groups work in HGABAC[7]. In this case Alice and Bob are members of the CS Faculty group

and inherit the attributes *role* and *department* with values "faculty" and "CompSci" respectively. Additionally, Bob is a member of the SoftEng Undergrad group and inherits the values "undergrad" and "SoftEng" for the attributes *role* and *department*. These inherited attributes are merged with the user's directly assigned attributes to form the user's effective attribute set (similar to how attributes are merged in Attribute Delegation). In Group Membership Delegation, membership in groups are delegated as opposed to the delegator's attributes. In Fig. 3, if Alice wanted to delegate a permission she was granted from belonging to the CS Faculty group (e.g. from satisfying the policy *"role = "faculty" AND department = "CompSci""*) to Dave she would delegate her membership in the CS Faculty group such that Dave's inherited set of attributes would be {*(role,* {*"undergrad", "faculty"*}*), (department,* {*"SoftEng", "CompSci"*}*)*} leading to the effective attribute set {*(year,* {*2*}*), (role,* {*"faculty", "undergrad"*}*), (department,* {*"CompSci", "SoftEng"*}*)*} when merged with his attributes.

This method of delegation has several advantages over Attribute Delegation. User comprehension is improved as users are not required to pick individual attributes to delegate and instead only need to consider what group memberships are needed. Placing constraints on delegation becomes easier as delegators are forced to delegate whole attribute sets belonging to groups at a time (constraints can be placed on what group memberships can be delegated and by whom, rather than individual attributes). Finally, the effective attribute set of delegatees is more likely to remain descriptive of the delegatee as personal attributes (like year, age, etc.) are more likely to be directly assigned than assigned to groups.

Despite these advantages, Group Membership strategies share a number of issues in common with Attribute Delegation. Conflicting policy evaluations and user collusion is still possible, although more restrained. For collusion to be possible, groups have to be assigned the required attribute value pairs. For example, if the policy was *"role = "faculty" AND department = "SoftEng""*, Alice and Dave could still collude to satisfy the policy (by Alice delegating her membership in the CS Faculty group to Dave); however, it would not be possible for Alice and Dave to collude to satisfy the policy *"year > 1 AND department = "CompSci""* as *year* is a directly assigned attribute. Isolating attribute sets obtained through membership delegation and attribute sets obtained through normal assignment

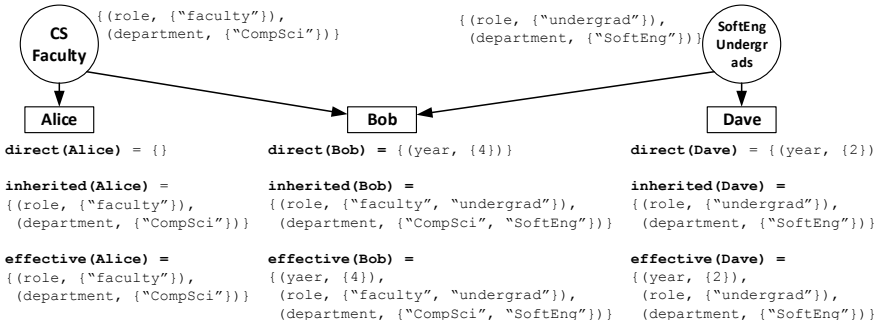

Fig. 3. Example of attribute user groups from HGABAC[7]. User groups are shown as circles and users as rectangles. Arrows denote a user being a member of a group.

would minimize the issue and avoid unforeseen permissions being granted (e.g. if Alice delegates her membership a group to Dave, she knows that Dave would not satisfy any policy that she her self could not satisfy from her membership).

Group Membership Delegation also introduces a new issue. Attributes that are directly assigned to a delegator, like the *year* attribute in Fig. 3, are undelegatable. Assuming this attribute is only directly assigned to users and never to groups, it would be impossible to delegate membership to satisfy a policy such as *"year* $\geq$ *2"*. A system utilizing Group Membership Delegation would either have to carefully design its groups such that all desired delegation use cases can be accomplished through delegating group memberships or implement a second delegation strategy in addition to Group Membership Delegation.

Permission Delegation Rather than delegating attributes (directly or indirectly) Permission Delegation strategies are based on delegating permissions. Delegators are able to delegate permissions they obtain by satisfying policies onto delegatees so long as the granting policy remains satisfied (e.g. if the delegator's attributes or an environmental attribute changes such that the policy granting the permission is no longer satisfied, the delegated permission is revoked). In strategies where a group is the delegator, the permissions the group can delegate is equal to the set of permissions a user would be granted if they had the same attributes as the group. For example, if the users and groups from Fig. 3 and the policy *"role = "faculty" AND department = "CompSci""* existed that granted the permission, p_1, both Alice and Bob as well as the group CS Faculty could delegate p_1. If the policy *"year* $\geq$ *2 AND TIME* $>$ *9:00AM AND TIME* $<$ *5:00PM"* granted the permission p_2, Bob and Dave could delegate p_2 but the delegation would only be valid between 9:00AM and 5:00PM.

Permission Delegation strategies poses greater challenges in terms of implementation but resolve the issues faced by the other families. As delegated permissions are only valid while the policy granting them remains satisfied, a system would be required to either periodically check that the delegator still satisfies the policy or recheck the policy each time the delegatee uses the permission. Depending on the size of the system and the complexity of the policies, this could add significant overhead. The benefit is that no change is made to the delegatee's attribute set, limiting conflicting policy evaluations and preventing user collusion. User comprehension is also improved as users are delegated permissions directly rather than attributes that only indirectly grant permissions.

3 Conclusions & Future Work

3.1 Delegation Strategies

The ideal delegation strategy depends on the needs of the implementing system; however, a few generalizations can be made. Permission Delegation is suitable for systems requiring high user comprehension and removes the possibility of conflicting policy evaluations and user collusion. Attribute Delegation is ideal when continual policy evaluation would be difficult or low implementation complexity is desired. Group Membership Delegation provides high user comprehension with similar results to Attribute Delegation but requires group support.

Delegating to a user (X-to-User strategies) provides the closest parallel to delegation in traditional models, however, delegating to groups (X-to-Group), attributes (X-to-Attribute) or policies (X-to-Policy) can provide greater flexibility and allow for delegation to users whose identity is unknown during policy creation. X-to-Group allows for delegation to groups of users in one operation but requires group support. X-to-Policy introduces higher revocation complexity and lower user comprehension but has the greatest flexibility. X-to-Attribute provides a middle ground between the two with less flexibility than X-to-Policy but increases user comprehension while retaining the identity-less nature of ABAC.

3.2 Future Work

A number of directions are possible for future work. Using multiple strategies simultaneously could provide new possibilities for delegation. Such combinations could help overcome the limitations of individual strategies but further work is needed to evaluate any complexities or conflicts introduced. Existing policy conflict resolution techniques could help mitigate the issues faced by Attribute and Group Membership Delegation, as well as allow for hybrid strategies with minimal limitations. Additional work is required to determine if current techniques are applicable. Formalizing the strategies described in this work will allow for in-depth analysis and aid integration into existing ABAC models. Extending an existing model with each strategy would allow for a more quantitative evaluation and provide a reference model for future work. HGABAC is an ideal candidate for such extensions by virtue of its support for user groups.

References

1. E. Barka, R. Sandhu, et al. A Role-Based Delegation Model and Some Extensions. In *NISSC'00*, pages 396–404, 2000.
2. J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-Policy Attribute-Based Encryption. In *SP'07*, pages 321–334, 2007.
3. K. Z. Bijon, R. Krishman, and R. Sandhu. Constraints Specification in Attribute Based Access Control. *Science*, 2(3):131–144, 2013.
4. D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST Standard for Role-Based Access Control. *TISSEC*, 4(3):224–274, 2001.
5. X. Jin, R. Krishnan, and R. S. Sandhu. A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC. *DBSec*, 12:41–55, 2012.
6. D. Servos, S. Mohammed, J. Fiaidhi, and T. Kim. Extensions to Ciphertext-Policy Attribute-Based Encryption to Support Distributed Environments. *IJCAT*, 47(2-3):215–226, 2013.
7. D. Servos and S. L. Osborn. HGABAC: Towards a Formal Model of Hierarchical Attribute-Based Access Control. In *FPS'14*, pages 187–204, 2014.
8. S. Turner, R. Housley, et al. An Internet Attribute Certificate Profile for Authorization. RFC 5755, January 2010.
9. H. Wang and S. L. Osborn. Static and Dynamic Delegation in the Role Graph Model. *IEEE TKDE*, 23(10):1569–1582, 2011.